A Neuro Genetic System Based Approach for enhancing Software Development Effort Estimation

A Research Proposal

For the Partial fulfillment of Degree of

Doctor of Philosophy

In

Computer Engineering

By

Poonam Rijwani

Under the Guidance of

Dr. Sonal Jain

Associate Professor

Department of Computer Engineering,

JK LAXMIPAT UNIVERSITY

Area of Research: SOFT COMPUTING

Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect, the role model for soft computing is the human mind. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost.

The principal constituents of Soft Computing (SC) are Fuzzy Logic (FL), Neural Computing (NC), Evolutionary Computation (EC) Machine Learning (ML) and Probabilistic Reasoning (PR), with the latter subsuming belief networks, chaos theory and parts of learning theory.

Topic of Research: A Neuro Genetic System Based Approach for enhancing Software Development Effort Estimation

In recent years, the developments of large scale software products gain a growing interest. For any industry to stay competitive, managing balance between quality and cost of software is important. Estimating software development effort remains a complex problem and one which continues to attract considerable research attention.

Accurate cost estimation is important because it can help to classify and prioritize development projects to determine what resources to commit to the project and how well these resources will be used. In the worst case, overrunning projects are cancelled and the entire development effort is wasted. For example, NASA cancelled its incomplete Check-out Launch control system project after the initial $200 M estimate was exceeded by another $200 M.

An extremely helpful form of effort prediction is the one made at an early stage during a project. However, estimates at the preliminary stages of the project are the most difficult to obtain because the primary source to estimate the costing comes from the requirement specification document. Due to the intangible nature of the product "Software", software developing companies are often faced with the problems estimating the effort needed to complete a software project.

Among the software cost estimation techniques, COCOMO (constructive cost model) is the most used algorithmic cost modeling technique because of its simplicity for estimating the effort in person months for a project at different stages.

Software cost estimation is the process of predicting the most realistic use of effort required to develop or maintain software. Effort estimates are used to calculate effort in person-months (PM) for the software development work elements of the work breakdown structure (WBS).

Accurate effort estimation is important because:

- It can help to classify and prioritize development projects with respect to an overall business plan.
- It can be used to determine what resources to commit.
- It can be used to assess the impact of changes and support re-planning.
- Projects can be easier to manage and control when resources are better matched to real needs.
- Customers expect actual development costs to be in line with estimated costs.

This paper proposes a neural network technique using back propagation learning algorithm for software cost estimation which is based on COCOMO model. Further, we'll apply genetic algorithm for topology optimization. GA is used to select a topology (number of hidden layers, number of hidden nodes, interconnection pattern) for the ANN which in turn is trained using some training scheme.

1. COCOMO Model

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics. **CO**nstructive **CO**st **MO**del II (COCOMO® II) allows one to estimate the cost, effort, and schedule when planning a new software development activity.

The Constructive Cost Model (COCOMO 81), a well-known cost and schedule estimation model, was originally published in the text *Software Engineering Economics.* The model was defined based on the analysis of 63 completed projects from different domains during the 1970s

and the early 1980s. To address those issues emerging from changes in technologies and development processes, the USC Center for Systems and Software Engineering has developed and published COCOMO II, a major extension to COCOMO 81. Among the main upgrades are the introduction of new functional forms that use scale factors, new cost drivers, and a set of parameters' values.

The original COCOMO® model was first published by Dr. Barry Boehm in 1981, and reflected the software development practices of the day. In the ensuing decade and a half, software development techniques changed dramatically. These changes included a move away from mainframe overnight batch processing to desktop-based real-time turnaround; a greatly increased emphasis on reusing existing software and building new systems using off-the-shelf software components; and spending as much effort to design and manage the software development process as was once spent creating the software product.

Boehm proposed three levels of the model: basic, intermediate, detailed.

- The basic COCOMO'81 model is a single-valued, static model that computes software development effort (and cost) as a function of program size expressed in estimated thousand delivered source instructions (KDSI).

- The intermediate COCOMO'81 model computes software development effort as a function of program size and a set of fifteen "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.

- The advanced or detailed COCOMO'81 model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

COCOMO'81 models depends on the two main equations

1. *Development effort* : $MM = a * KDSI\ b$

based on MM - man-month / person month / staff-month is one month of effort by one person. In COCOMO'81, there are 152 hours per Person month. According to organization this values may differ from the standard by 10% to 20%.

2. *Effort and development time* (TDEV): $TDEV = 2.5 * MM\ c$

The coefficients a, b and c depend on the *mode of the development.* There are three modes of development:

Development Mode	Project Characteristics			
	Size	Innovation	Deadline/constraints	Dev. Environment
Organic	Small	Little	Not tight	Stable
Semi-detached	Medium	Medium	Medium	Medium
Embedded	Large	Greater	Tight	Complex hardware/ customer interfaces

Basic COCOMO

The basic COCOMO applies the parameterised equation without much detailed consideration of project characteristics.

$MM = a * KDSI^{b}$
$TDEV=2.5 * MM^{c}$

Basic COCOMO	a	b	c
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Intermediate COCOMO

The same basic equation for the model is used, but fifteen cost drivers are rated on a scale of 'very low' to 'very high' to calculate the specific effort multiplier and each of them returns an adjustment factor which multiplied yields in the total EAF (Effort Adjustment Factor). The adjustment factor is 1 for a cost driver that's judged as normal.

In addition to the EAF, the model parameter "a" is slightly different in Intermediate COCOMO from the basic model. The parameter "b" remains the same in both models.

$MM = a * KDSI^{b}$
$TDEV = 2.5 * MM^{c}$

Intermediate COCOMO	a	b	c
Organic	3.2	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	2.8	1.20	0.32

Man Month Correction is now:

$$MM_{Korr} = EAF * MM_{nominal}$$

Advanced, Detailed COCOMO

The Advanced COCOMO model computes effort as a function of program size and a set of cost drivers weighted according to each phase of the software lifecycle. The Advanced model applies the Intermediate model at the component level, and then a phase-based approach is used to consolidate the estimate [Fenton, 1997]. The four phases used in the detailed COCOMO model are: requirements planning and product design (RPD), detailed design (DD), code and unit test (CUT), and integration and test (IT). Each cost driver is broken down by phases as in the example shown in Table 1. Estimates for each module are combined into subsystems and eventually an overall project estimate. Using the detailed cost drivers, an estimate is determined for each phase of the lifecycle.

Cost Driver	Rating	RPD	DD	CUT	IT
ACAP	Very Low	1.80	1.35	1.35	1.50
	Low	0.85	0.85	0.85	1.20
	Nominal	1.00	1.00	1.00	1.00
	High	0.75	0.90	0.90	0.85
	Very High	0.55	0.75	0.75	0.70

Table 1: Analyst capability effort multiplier for detailed COCOMO

Differences Between COCOMO I and COCOMO II

The major differences between COCOMO I AND COCOMO II are:

(1) COCOMO'81 requires software size in KDSI as an input, but COCOMO II is based on KSLOC (logical code). The major difference between DSI and SLOC is that a single Source Line of Code may be several physical lines. For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI.

(2) COCOMO II addresses the following three phases of the spiral life cycle: applications development, early design and post architecture COCOMO'81 provides point estimates of effort and schedule, but COCOMO II provides likely ranges of estimates that represent one standard deviation around the most likely estimate.

(3) The estimation equation exponent is determined by five scale factors (instead of the three development modes) Changes in cost drivers are:

- Added cost drivers (7): DOCU, RUSE, PVOL, PLEX, LTEX, PCON, SITE
- Deleted cost drivers (5): VIRT, TURN, VEXP, LEXP, MODP
- Alter the retained ratings to reflect more up-do-date software practices
- Data points in COCOMO I: 63 and COCOMO II: 161
- COCOMO II adjusts for software reuse and reengineering where automated tools are used for translation of existing software, but COCOMO'81 made little accommodation for these factors
- COCOMO II accounts for requirements volatility in its estimates

Table 1. COCOMO 81 cost driver values

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	1.46	1.19	1.00	0.86	0.71	
PCAP	1.42	1.17	1.00	0.86	0.70	
AEXP	1.29	1.13	1.00	0.91	0.82	
MODP	1.24	1.10	1.00	0.91	0.82	
TOOL	1.24	1.10	1.00	0.91	0.83	
VEXP	1.21	1.10	1.00	0.90		
LEXP	1.14	1.07	1.00	0.95		
DATA		0.94	1.00	1.08	1.16	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
TURN		0.87	1.00	1.07	1.15	
VIRT		0.87	1.00	1.15	1.30	
STOR			1.00	1.06	1.21	1.56
TIME			1.00	1.11	1.30	1.66
RELY	0.75	0.88	1.00	1.15	1.40	
SCED	1.23	1.08	1.00	1.04	1.10	

Table 2. COCOMO II cost driver values

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00
RELY	0.82	0.92	1.00	1.10	1.26	
DATA		0.90	1.00	1.14	1.28	
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	
TIME			1.00	1.11	1.29	1.63
STOR			1.00	1.05	1.17	1.46
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.42	1.19	1.00	0.85	0.71	
PCAP	1.34	1.15	1.00	0.88	0.76	
PCON	1.29	1.12	1.00	0.90	0.81	
AEXP	1.22	1.10	1.00	0.88	0.81	
PEXP	1.19	1.09	1.00	0.91	0.85	
LTEX	1.20	1.09	1.00	0.91	0.84	
TOOL	1.17	1.09	1.00	0.90	0.78	
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	

COCOMO 81:

$$\log(PM) = \beta_0 + \beta_1 \log(Size) + \beta_2 \log(EM_1) + \ldots + \beta_{16} \log(EM_{15})$$

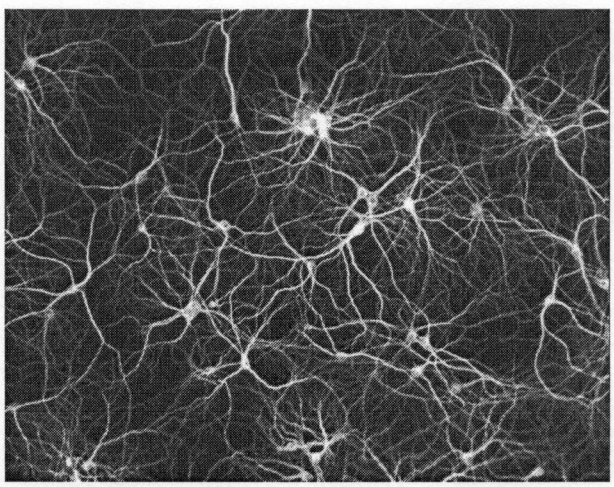
COCOMO II:

$$\log(PM) = \beta_0 + \beta_1 \log(Size) + \beta_2 SF_1 \log(Size) + \ldots + \beta_6 SF_5 \log(Size) + \beta_7 \log(EM_1) + \ldots + \beta_{23} \log(EM_{17})$$

If in the context of discussing COCOMO® these terms are used: Basic, Intermediate, or Detailed for model names; Organic, Semidetached, or Embedded for *development mode*, then the model being discussed is COCOMO® 81. However, if the model names mentioned are Application Composition, Early Design, or Post-architecture; or if there is mention of *scale factors* Precedentedness (PREC), Development Flexibility (FLEX), Architecture/Risk Resolution (RESL), Team Cohesion (TEAM), or Process Maturity (PMAT), then the model is COCOMO® II.

2. Artificial Neural Network Concepts

Human brain has excellent computational and logical power. An ANN is an imitation of biological human brain. Neural Network is just a web of inter connected neurons which are millions and millions in number. With the help of these interconnected neurons all the parallel processing is done in human body and the human body is the best example of Parallel Processing.



An Artificial Neuron is basically an engineering approach of biological neuron. It has device with many inputs and one output. ANN consists of large number of simple processing elements that are interconnected with each other and layered also. The biological brain has two fundamental components: first neurons, which becomes nodes of ANN and second, synapses, which becomes weight or connections of ANN.

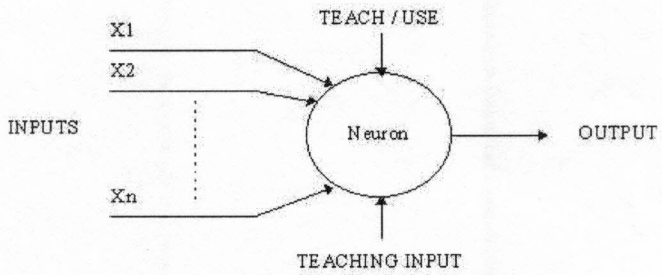
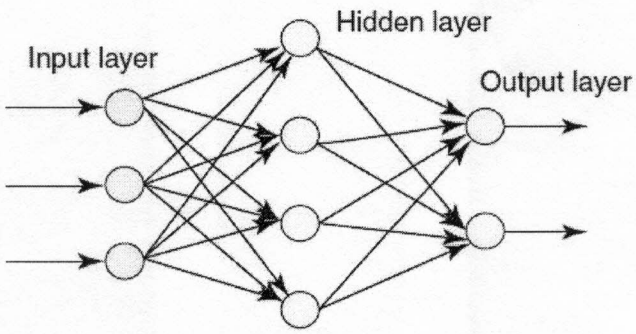


Fig: Artificial Neuron



(b) Multilayered artificial neural network

Artificial neural network is a promising technique to build predictive models, because they are capable of modeling non linear relationships. ANNs possess large number of highly interconnected processing elements called neurons, which usually operate in parallel and are configured in regular architectures.

The neuron computes a weighted sum of its input and generates an output if the sum exceeds a certain threshold. This output then becomes an excitatory (positive) or inhibitory (negative) input to other neurons in the network. The process continues until one or more outputs are generated.

The ANN is initialized with random weights and gradually learns the relationships implicit in a training dataset by adjusting its weights when presented to these data.

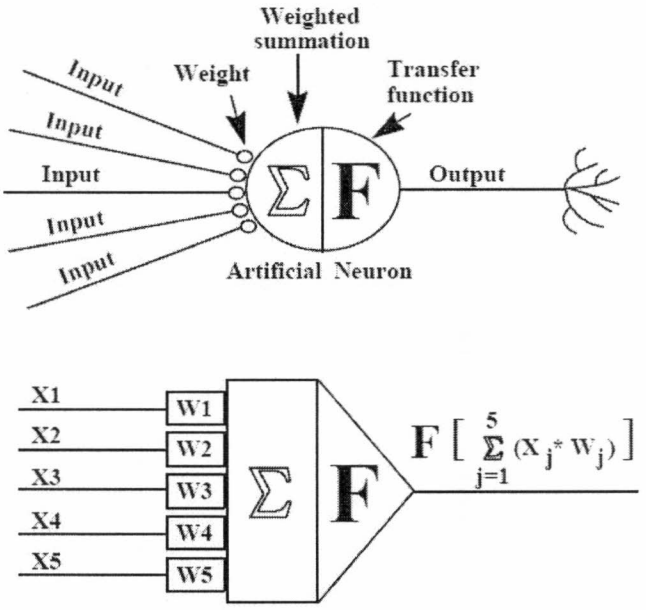
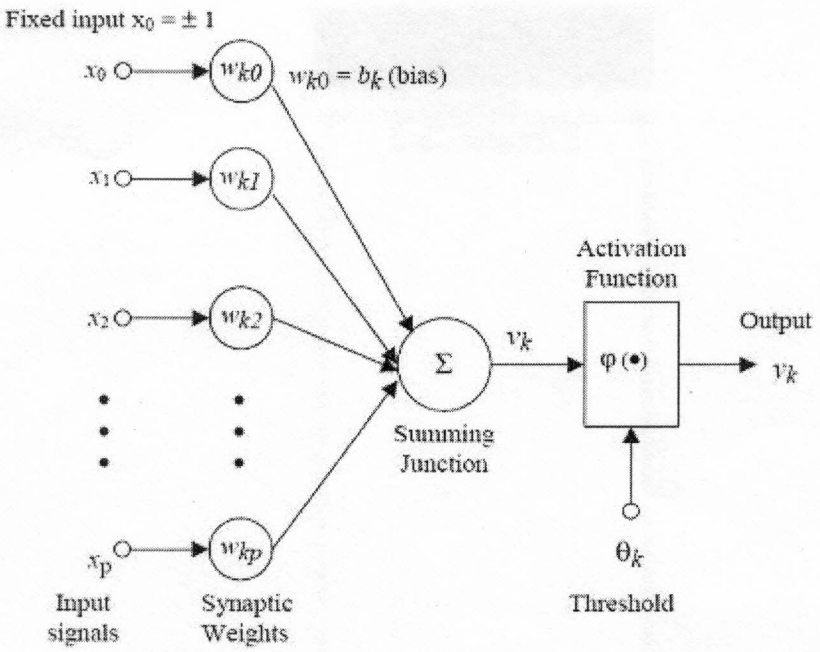


Fig: function of an Artificial Neuron

The Mathematical Model

Once modeling an artificial functional model from the biological neuron, we must take into account three basic components. First off, the synapses of the biological neuron are modeled as weights. Let's remember that the synapse of the biological neuron is the one which interconnects the neural network and gives the strength of the connection. For an artificial neuron, the weight is a number, and represents the synapse. A negative weight reflects an inhibitory connection, while positive values designate excitatory connections. The following components of the model represent the actual activity of the neuron cell. All inputs are summed altogether and modified by the weights. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

Mathematically, this process is described in the figure



From this model the interval activity of the neuron can be shown to be:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

The output of the neuron, Vk, would therefore be the outcome of some activation function on the value of vk.

Activation Functions

As mentioned previously, the activation function acts as a squashing function, such that the output of a neuron in a neural network is between certain values (usually 0 and 1, or -1 and 1). In general, there are three types of activation functions, denoted by Φ(.) . First, there is the Threshold Function which takes on a value of 0 if the summed input is less than a certain threshold value (v), and the value 1 if the summed input is greater than or equal to the threshold value.

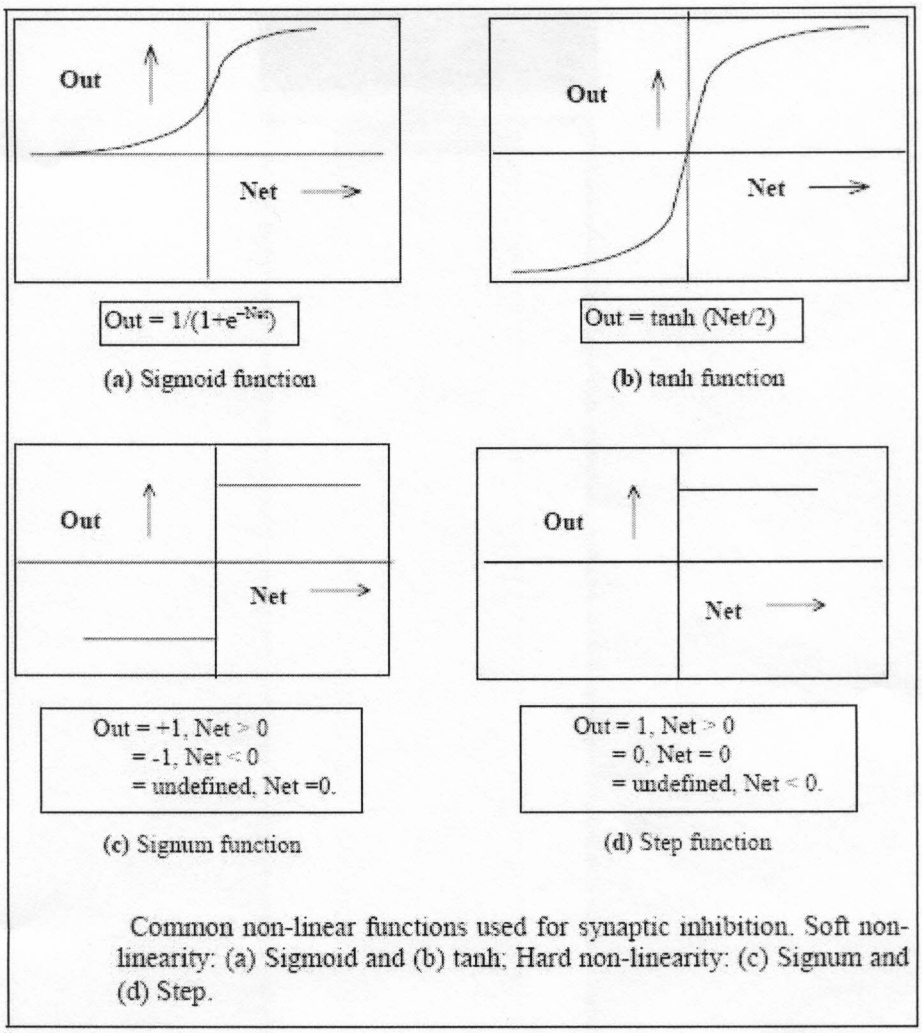
$$\varphi(v)=\begin{cases}1 & \text{if } v \geq 0\\ 0 & \text{if } v < 0\end{cases}$$

Secondly, there is the Piecewise-Linear function. This function again can take on the values of 0 or 1, but can also take on values between that depending on the amplification factor in a certain region of linear operation.

$$\varphi(v)=\begin{cases}1 & v \geq \frac{1}{2}\\ v & -\frac{1}{2} > v > \frac{1}{2}\\ 0 & v \leq -\frac{1}{2}\end{cases}$$

Thirdly, there is the sigmoid function. This function can range between 0 and 1, but it is also sometimes useful to use the -1 to 1 range. An example of the sigmoid function is the hyperbolic tangent function.

$$\varphi(v)=\tanh\left(\frac{v}{2}\right)=\frac{1-\exp(-v)}{1+\exp(-v)}$$



Common non-linear functions used for synaptic inhibition. Soft non-linearity: (a) Sigmoid and (b) tanh; Hard non-linearity: (c) Signum and (d) Step.

Neural Network Topologies

Here, we focus on the pattern of connections between the units and the propagation of data. As for this pattern of connections, the main distinction we can make is between:

- **Feed-forward neural networks**, where the data from input to output units is strictly feed forward. The data processing can extend over multiple (layers of) units, but no feedback connections are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers.
- **Recurrent Neural Networks** that do contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases,

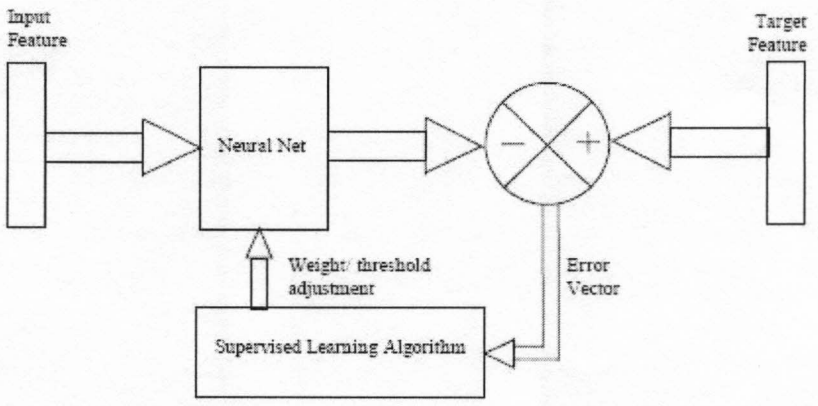
the activation values of the units undergo a relaxation process such that the neural network will evolve to a stable state in which these activations do not change anymore. In other applications, the change of the activation values of the output neurons are significant, such that the dynamical behaviour constitutes the output of the neural network (Pearlmutter, 1990).

Training of Artificial Neural Networks

A **neural network** has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to **'train' the neural network** by feeding it teaching patterns and letting it change its weights according to some learning rule.

We can categorize the learning situations in two distinct sorts. These are:

- **Supervised learning** or Associative learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the neural network (self-supervised).



- **Unsupervised learning** or Self-organisation in which an (output) unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

- **Reinforcement Learning** This type of learning may be considered as an intermediate form of the above two types of learning. Here the learning machine does some action on the environment and gets a feedback response from the environment. The learning system grades its action good (rewarding) or bad (punishable) based on the environmental response and accordingly adjusts its parameters.

  Generally, parameter adjustment is continued until an equilibrium state occurs, following which there will be no more changes in its parameters. The self organizing neural learning may be categorized under this type of learning.

3. Genetic Algorithm—a Global Optimizer

GA is based on biological evolutionary theories to solve optimization problems. GA comprises of a set of individual elements (the population) and a set of biologically inspired operators. According to evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring, and transmit their biological heredity to the new generations. GAs are much superior to conventional search and optimization techniques in high-dimensional problem spaces due their inherent parallelism and directed stochastic search implemented by recombination operators.

GA operates through a simple cycle of three stages:

1. Randomly create an initial population of individuals.
2. Perform the following sub steps iteratively for each generation until a termination condition is fulfilled:

2.1. Evaluate the fitness of each individual in the population and save the best individual of all preceding populations.

2.2. Create a new population by applying the genetic operators:

2.2.1. Selection;

2.2.2. Crossover;

2.2.3. Mutation;

2.3. Replace the current population by the new population.

3. Output the individual with the best fitness as the optimum solution.

Selection is based on fitness, i.e. the fitter an individual the greater the chance for this individual to get selected for reproduction and contribute offspring for the next generation.

Crossover operator takes two chromosomes and swaps part of their genetic information to produce new chromosomes.

Mutation is implemented by occasionally altering a random bit in a string before the off springs are inserted into the new population.

Control parameters: We can visualize the functioning of GAs as a balanced combination of *exploration* of new regions in the search space and *exploitation* of already sampled regions. The balance, which critically controls the performance of GAs is determined by the right choice of control parameters: the crossover and mutation probabilities and population sizes. The trade-offs that arise are:

- Increasing the crossover probability increases the recombination of building blocks, but it also increases the disruption of good strings.
- Increasing the mutation probability tends to transform the genetic search into a random search, but it also helps reintroduce lost genetic material.
- Increasing the population size increases its diversity and reduces the probability that the GA will prematurely converge to a local optimum, but it also increases the time required for the population to converge to the optimal regions in the search space.

4. Literature Survey

In the field of software effort estimation, the effort required to develop a new software project is estimated by taking the details of the new project into account. The specific project is then compared to a historical data set containing measurements of relevant metrics (e.g. size, language used, and experience of development team) and the associated development effort.

The first approaches to estimate software development effort were introduced in the late 1960s, and relied on expert judgment. In these cases, a domain expert applies his/her prior experience to come up with an estimation of the needed effort. A number of different variations exist, e.g., Delphi expert estimation, in which several experienced developers formulate an independent estimate and the median of these estimates is used as the final effort estimation. while still widely used in companies, an expert driven approach has the disadvantage of lacking an objective underpinning.

During the last 30 years, a number of formal models for software effort estimation have been proposed such as Cocomo, Cocomo II, SLIM, and Function Points Analysis. These models have some advantages, providing a formulaic underpinning of software effort estimation. Hence, these models allow for a number of analyses to be performed upon the obtained results. Companies applying formal models during the estimation process often opt for a Constructive Cost Model (Cocomo model).

Data for this model are collected making use of specific questionnaires which are filled in by the project manager. This data collection approach requires a considerable effort from the business. Also, it should be noted that the Cocomo I model is already somewhat outdated as, e.g., new software development trends such as outsourcing and multiplatform development are not taken into account by the model. A newer version of the Cocomo model exists, but the data on which this model was built is not publicly available.

More recently, formal models are being superseded by a number of data intensive techniques originating from the data mining literature .These include various regression techniques which result in a linear model, nonlinear approaches like neural networks, tree/rule based models such as CART, and lazy learning strategies (also referred to as case-based reasoning (CBR)).

A non exhaustive overview of the literature concerning the use of various machine learning approaches for software effort estimation is presented below. The table summarizes the applied modeling techniques, the data sets that are used, and the empirical setup for a number of studies.

Many researchers used their different ANN and different dataset, to predict the effort more correctly. G. E. Wittig, et al. used a dataset of 15 commercial systems, and used feed-forward back propagation multilayer neural network for their experiment. ANN used in this paper are with numbers of hidden layers varying from 1-6, but found the best performance for only one hidden layer with sigmoid function. It has been observed that for smaller system the error was 1% and for larger systems error was 14.2% of the actual effort. In a paper by Ali Idri, et al. uses COCOMO-81 dataset and three layered back-propagation ANN, applying 13 *cost*

drivers as inputs and *development effort* taken as output. The ANN used are with 13 neurons in hidden layer and experimented for 300,000 iterations to find the average MRE = 1.50%.

F. Barcelos Tronto, et al., also used COCOMO-81 dataset, with only one input, i.e TOTKDSI (thousands of delivered source instructions). All the input data were normalized to [0, 1] range. Here a feed-forward multilayer back-propagation ANN was used with the 1-9-4-1 architecture. The performance in MMRE found was 420, where as that of COCOMO and FPA was 610 and 103 respectively.

Jaswinder Kaur, et al. implemented a back-propagation ANN of 2-2-1 architecture on NASA dataset consist of 18 projects. Input was KDLOC and development methodology and effort was the output. He got result MMRE as 11.78.

Roheet Bhatnagar, et al. used MATLAB NN toolbox for effort prediction. He had used a dataset proposed by Lopez-Martin, which consists of 41 projects data. He has designed a 3-3-1 neural network, applied the Dhama Coupling (DC), McCabe Complexity (MC) and Lines of Code (LOC) as inputs. Development time was the only one output. The results of the experiment indicate that the percentage of error during training, validation and testing was between +14.05 to -25.60, +12.76 to -18.89 and +13.66 to -15.75 respectively.

K.K. Aggarwal , et al.had investigated for finding the best training algorithm. Here ISBSG repository data was used on a 4-151 feed-forward ANN. Four inputs were taken-FP, FP standard, language and maximum team size. SLOC was the only output. The various training algorithm for ANN has been used and concluded that =trainbr' is the best algorithm. =traingd' was found to be the next best algorithm.

Table 3: Dataset and ANN used

Author	Learning Algorithm	Dataset	No. of Projects	No. of Inputs	ANN Configurations
I.F. Barcelos Tronto	Backpropagation	COCOMO	63	1	[1-9-4-1]
G. E. Wittig	Back-Propagation	Commercial Systems	15	-	[23-4-1]
Ali Idri	Back-Propagation	Commercial Systems	63	13	[13-13-1]
Jaswinder Kaur	Back-propagation	NASA	18	2	[2-2-1]
Mrinal KantiGhose	Backpropagation	LopezMartin	41	3	[3-3-1]
A.R. Venkatachalam	Back-propagation	COCOMO	63	22	[22-45-2]

Table 4: Overview of the Application of various Approaches for Software Effort Estimation

Author	**Title & Journal**	**Year**	**What?**	**Techniques**	**Dataset**	**Metrics-Empirical setup**
k.Srinavasan, D.Fisher	Machine learning approaches to estimating software development effort. *IEEE transactions on Software Engineering*	1995	Two alternative data mining technique are compared to formal models	Artificial Neural Networks, CART	Kemerer 15 projects-6 attributes COCOMO81 63 projects-16 attributes	MMRE, R2 – Holdout
M.Shepherd, C.Schofield	Estimating software project effort using Analogies *IEEE transactions on Software Engineering*	1997	Investigation of data mining techniques as an alternative to formal models	OLS regression, case based reasoning	DPS database 24proj.- 5 attributes Desharnais 77 proj.- 9 attributes Finnish 38 proj.-29 attributes Kemerer 15	MMRE, pred25-cross validation

					projects-2 attributes Mermaid 28 proj-17 attributes Telecom datasets 18-33 projects- 1-13 attribues	
G.Wittig, G.Finnie	Estimating software development effort with connectionist models *Information and software Technology*	1997	Analysis of Back propagation neural networks for software effort prediction	Artificial Neural Networks	Simulated data 1000 proj. -3 attributes Desharnais 81 proj.- 9 attributes	MMRE, pred25 Holdout
G.R.Finnie, G.E.Wittig, J.M. Desharnais	A comparison of software Effort Estimation techniques: Using Function Points with Neural networks, Case based reasoning and Regression models. *The journal of Systems and*	1997	Comparison of three different estimation techniques using function points as a size estimate	OLS regression, artificial neural networks, case based reasoning	ASMA data set 299 proj. -1 attribute	MMRE, pred25 Holdout

	Software.					
C.Burgess, M.Lefley	Can genetic programming improve software effort estimation? A comparative evaluation *Information and Software Technology*	2001	Evaluation of genetic programming algorithms for software effort prediction	Artificial neural networks, case based reasoning, genetic programming	Desharnais 81 proj.- 9 attributes	MMRE,BMMRE, Pred25, co-relation, AMSE Holdout
B. Kitchenham, S.L. Pfleegar, B.McColl, S.Eagan	An empirical study on maintainance and development accuracy *The journal of Systems and Software*	2002	Comparison of regression techniques using a real-life data set from a single company	OLS regression, median regression	CSC dataset 144 proj. 10 attributes	MMRE, pred-Holdout,and paired t-test
P.sentas, L.Angelis, I.stamelos, G.Bleris	Software productivity and effort prediction with ordinal regression *Information and Software Technology*	2005	Analysis of a novel regression technique for software effort prediction	OLS regression, ordinal regression	COCOMO81 63 proj.-22 attributes, Maxwell 62 proj, -14 attributes, ISBSG R7 52 proj. -10 attributes	MMRE, Pred25 Holdout
T.Menzies,	Selecting best practices for	2006	Investigation of	OLS regression.	COCOMO81 63	MMRE, Pred30,

Z.Chen, J. Hihn, K.Lum	Effort estimation *IEEE transactions on software Engineering*		deviations, exhibited by different techniques for software estimation		proj.-16 attributes, NASA 93 proj. 16 attributes, COCOMOII 161 proj. 18 atrributes.	Corelation-Holout
I.F.Barcelos C.postal	The artificial neural networks model for software effort estimation *International Network of Philosophers of Education*	2006	Neural network compared with conventional regression analysis	Artificial Neural Networks, Regression	COCOMO 63 proj. 16 attributes	MMRE
N.-H. Chiu, S.-J. Huang	The adjusted analogy based softwre effort estimation based on similarity distances The journal of Systems and Softwares	2007	The application of a genetic algorithm to derive an estimation based on the retrieved cases	OLS regression,artificial neural networks, CART, case-based reasoning	DPS database 23 proj. -5 attributes. Abran 21 proj.- 6 attributes	MMRE, MdMRE,pred25, cross validation

K.Kumar, V.Ravi, M.Carr, N.kiran	Software development cost estimation using wavelet neural networks The journal of System & Softwares	2008	Investigation of wavelet neural networks in the context of software effort estimation	Artificial neural networks, RBF networks, wavelet neural networks, support vector machines	DPS database 23 proj. 4 attributes. Abran 21 proj. 6 attributes	MMRE, MdMRE,pred25, cross validation
H.Park, S. Baek	An empirical validation of a neural network model for software effort estimation Expert Systems with Applications	2009	The evaluation of a neural network model for software effort estimation for function point based datasets	OLS regression, artificial neural networks	Korean IT dataset 148 proj 18 attributes	MMRE- Holout
Ch. Satyananda Reddy, KYSVN Raju	A concise Neural Network model for estimating software effort. International journal of recent trends in engineering	2010	Cost estimation model based on multi layer feed forward neural networks	Feed forward multilayer perceptron with back propogation and sigmoid activation function	COCOMOII	MRE

5. Proposed Model

In despite of extensive research over the last 20 years, the software community is still significantly challenged when it comes to effective resource prediction. The performance of a neural network depends on its architecture and their parameter settings. Their are many parameters governing the architecture of the neural network including the number of layers, the number of nodes in each layer , the transfer function in each node , learning algorithm parameters and the weights which determine the connectivity between nodes. There is no rule which determines the ideal parameter settings but even a slight parameter settings changes can cause major variations in the results of almost all networks. GA will be used for topology optimization. The proposed model uses single layer feed forward network architecture with back propagation.

6. Methodology

Design: The present research uses Experimental Research Design.

An attempt by the researcher to maintain control over all factors that may affect the result of an experiment. In doing this, the researcher attempts to determine or predict what may occur.

Dataset Preparation

The most important step in software cost estimation using neural network is the dataset preparation. COCOMO 81 dataset is publicly available and is converted into COCOMOII dataset . For this conversion, we need to convert the exponent, the size estimate and the ratings for the cost drivers. The following steps were employed to make the conversion:

• **Update size:** Depending upon language used for that project, each and every project's size is modified. Language information is given in COCOMO81 dataset which consists of 63 software projects written in Fortran, machine languages, Cobol, jovial, pl/I, higher languages, Pascal.

• **Convert exponent:** Conversion of COCOMO81 modes into COCOMOII Scale Factors is done using the type of the project (organic, semi -Detached and embedded). An exception is the process Maturity (PMAT) scale factor, which replaces the COCOMO 81 Modern Programming Practices (MODP) cost driver.

• **Rate Cost drivers**: The 10 cost drivers which exist in both COCOMO81 and COCOMOII (RELY, DATA, CPLX, TIME, STOR, ACAP, PCAP, AEXP, TOOL, and SCED) are kept same. For remaining the following conversion is used. For COCOMOII drivers PVOL, PEXP, LTEX ratings Of COCOMO81 drivers VIRT, VEXP, LEXP are used respectively. RUSE and PCON are set to nominal and SITE is set to high. Rating of DOCU Is set depending upon the type of the project. Rating of PMAT is adjusted depending upon the rating of MODP cost driver of COCOMO81.

ANN Preparation

Proposed Algorithm

Prior to applying the proposed algorithm, the neural network is trained in four stages by using feed forward back propagation algorithm for the experimental Data (COCOMOII dataset).

The steps involved in the algorithm are as follows:

(1) Initialization of weights

Step 1:-Initialize weights to small random Values.

Step 2:-While stopping condition is false, do steps 3 – 10.

Step 3:-For each training pair do steps 4-9.

(2) Feed Forward

Step 4:-Each input unit receives the input signal xi and transmits this signals to all units in the layer above i.e. Hidden units

Step 5:-Each hidden unit (zj , j=1,.....,p) sums its weighted input signals, Z-inj

$$z_{-inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

Next, the BINARY SIGMOIDAL function z = f(z) j in j is applied to all units in the layer above i.e. output units.

Step 6:-Each output unit (yk , k=1,……..,m) sums its weighted input signals,

$$y_{-ink} = w_{ok} + \sum_{j=1}^{n} z_j w_{jk}$$

and applies its BINARY SIGMOIDAL function to calculate the output signals.

$$Y = f(y_{-ink})$$

(3) Back Propagation of errors

Step 7:-Each output unit(yk , k=1,……,m) receives a target pattern corresponding to an input pattern, error information term is calculated as

$$\delta_k = (t_k - y_k) f(y_{-ink})$$

Step 8:-Each hidden unit (zj , j=1,……...,n) sums its delta inputs from units in the layer above

$$\delta_{-ink} = \sum_{k=1}^{m} \delta_j w_{jk}$$

The error information term is calculated as

$$\delta_j = \delta_{-ink} f(z_{-inj})$$

(4) Updation of weight and biases

Step 9:-Each output unit (yk , k=1,…….,m) updates its bias and weights (j=0,…….,p)

The weight correction term is given by

$$\Delta W_{jk} = \alpha \delta_k z_j$$

And the bias correction term is given by

$$\Delta W_{ok} = \alpha \delta_k$$

Therefore,

$$\Delta W_{jk}(new) = \Delta W_{jk}(old) + \Delta W_{jk}$$
$$\Delta W_{ok}(new) = \Delta W_{ok}(old) + \Delta W_{ok}$$

Each hidden unit (zj , j=1,.....,p) updates its bias and weights (i=0, ...,n). The weight correction term

$$\Delta V_{ij} = \alpha \delta_j x_i$$

And bias correction term

$$\Delta V_{oj} = \alpha \delta_j$$

Therefore,

$$\Delta V_{ij}(new) = \Delta V_{ij}(old) + \Delta V_{ij}$$
$$\Delta V_{oj}(new) = \Delta V_{oj}(old) + \Delta V_{oj}$$

Step 10:- Test the stopping condition

All the networks will be trained for 10 iterations and the most accurate results were considered.

Evaluation Method

Evaluation Criteria

This section presents the methodology used in conducting experiments and discusses the results obtain when applying our proposed neural network to the COCOMO dataset. The analysis undertaken in this study and the dataset used in this work are from COCOMO database, a dataset publicly available which consists of 63 projects. We have divided the entire dataset into two sets, training set and validation set in the ratio of 80%: 20%. Training set consists of 50 projects selected randomly and validation set consists of remaining 13 projects. All the data is normalized to fit in the natural logarithmic space.

The following evaluation criterion is used to assess and compare the performance of the neural network model. A common criterion for the evaluation of cost estimation model is the magnitude of relative error (MRE), and mean magnitude of relative error (MMRE). MRE is defined as:

$$MRE = \frac{|ActuralEffort - \Pr edictedEffort|}{ActualEffort} \times 100 \quad (1)$$

And Mean Magnitude of Relative Error (MMRE) for N projects is defined as in:

$$MMRE = \frac{1}{N} \sum_{i=1}^{N} MRE_i$$

For both MRE and MMRE, a higher score means worse prediction accuracy. When using MRE as a measure of prediction accuracy, we suppose the error is proportional to the size of the project. Thus, PRED (p) is usually used as a complementary criterion. This is defined as:

$$PRED(p) = \frac{K}{N} \times 100$$

Where k is the number of projects where MRE is less than or equal to p. Unlike both MRE and MMRE, for PRED(p), a higher score implies better prediction accuracy. In evaluating the trained network model, the MRE values are calculated using (1) for each project.

7. References

1. Donald J. Reifer, Barry W. Boehm, Sunita Chulani, "The Rosetta Stone:Making COCOMO 81 Files Work With COCOMO II", ROSSTALK, The Journal of Defense Software Engineering, February, 1999
2. B.Tirimula Rao , B.Sameet , G.Kiran Swathi , K.Vikram Gupta , Ch.Ravi Teja , S.Sumana "A Novel Neural Network Approach For Software Cost Estimation Using Functional Link Artificial Neural Network (FLANN).IJCSNS International Journal Of Computer Science And Network Security,Vol.9No.6,June2009.
3. I.F. Barcelos Tronto, J.D. Simoes da Silva, N. Sant. Anna , —Comparison of Artificial Neural Network and Regression Models in Software Effort Estimation‖, INPE ePrint, Vol.1, 2006.
4. Simon Haykin, —Neural Networks: A Comprehensive Foundation‖, Second Edition, Prentice Hall, 1998.
5. Ali Idri and Taghi M. Khoshgoftaar& Alain Abran,‖Can Neural Networks be easily Interpreted in Software Cost Estimation‖, IEEE Transaction, 2002, page:1162-1167.
6. G.E. Wittig and G.R Finnic, —Using Artificial Neural Networks and Function Points to estimate 4GL software development effort‖, AJIS,1994, page:87-94.
7. JaswinderKaur, Satwinder Singh, Dr. Karanjeet Singh Kahlon, PourushBassi,—Neural Network-A Novel Technique for Software Effort Estimation‖, International Journal of

Computer Theory and Engineering, Vol. 2, No. 1 February, 2010, page:17-19.

8. Roheet Bhatnagar, Vandana Bhattacharjee and Mrinal Kanti Ghose, —Software Development Effort Estimation –Neural Network Vs. Regression Modeling Approach‖, International Journal of Engineering Science and Technology,Vol. 2(7), 2010,page: 2950-2956.
9. K.K. Aggarwal, Yogesh Singh, Pravin Chandra and Manimala Puri, —Evaluation of various training algorithms in a neural network model for software engineering applications‖ , ACM SIGSOFT Software Engineering , July 2005, Volume 30 Number 4 , page: 1-4.
10. Mrinal Kanti Ghose, Roheet Bhatnagar and Vandana Bhattacharjee , —Comparing Some Neural Network Models for Software Development Effort Prediction‖ , IEEE, 2011.
11. B.W. Boehm, —*Software Engineering Economics*‖, Prentice-Hall, Englewood Cliffs, NJ, 1981.
12. B.W. Boehm, E. Horowitz, R. Madachy, et al., —*Software Cost Estimation with COCOMO II*‖, Prentice-Hall, Upper Saddle River, NJ, USA, 2000
13. Dawson, C.W., "A neural network approach to software projects effort estimation," Transaction: Information and Communication Technologies, Vol. 16, pages 9, 1996.
14. U.K. Chakraborty, A Branching Process Model for Genetic Algorithms, Information Processing Letters, 46, Elsevier, Amsterdam, 1995, pp. 281–292.
15. R. Salomon, Evolutionary algorithms and gradient search: similarities and differences, IEEE Transactions on Evolutionary Computation 2 (2) (1998) 45–55.
16. C. J. Burgess, M. Lefley, (2005), "Can Genetic Programming improve Software Effort Estimation? A Comparative Evaluation", Machine Learning Applications In Software Engineering: Series on Software Engineering and Knowledge Engineering, pp. 95–105.